METHOD/DEVICE FOR COMPILING AND METHOD/DEVICE FOR CALCULATING METHOD ACTIVITY DEGREE

Patent number: JP2000222221
Publication date: 2000-08-11
Inventor: OGASAWARA TAKESHI
Applicant: INTERNATL BUSINESS MACH CORP < IBM>
Classification:
- international: G06F9/45; G06F9/42; G06F11/34
- european:
Application number: JP19990022030 19990129
Priority number(s):

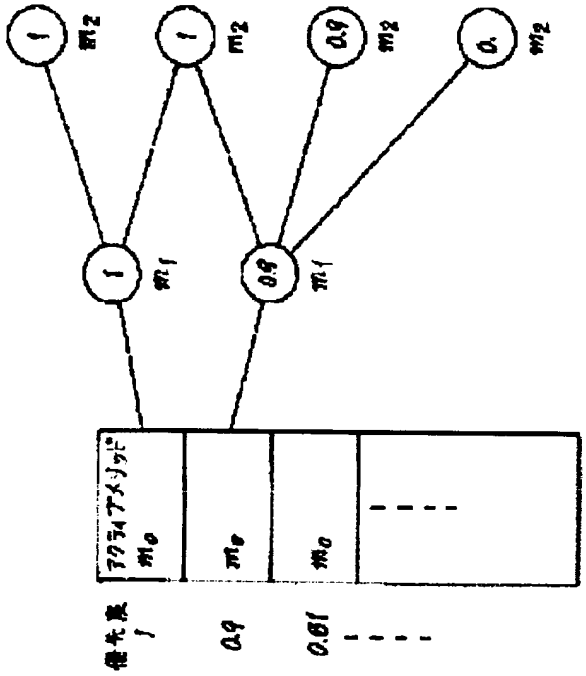
Also published as:
US6671877 (B1)

Report a data error here

Abstract of **JP2000222221**

PROBLEM TO BE SOLVED: To provide a method for calculating a method activity degree for effectively selecting a method for destructing a code.

SOLUTION: In compiling, a calling map concerning method calling which can be generated in the method is prepared and stored in a storing device, and a code for recording method calling generated actually is generated and stored in the storing device. In recovering the code in a state like this, all the threads are temporarily stopped to calculate activity degree expressing high or low possibility that some method is to be executed. Through the use of the calling map and information concerning method calling generated actually concerning a first method corresponding to a stuck frame, a second method of high possibility of being called from the first method is specified and stored in the storing device and priority according to the number of stages from the highest part of the stuck of the stuck frame corresponding to the first method is operated to the activity degree of the second method to update the activity degree of the second method. It is suitable to repeat processing like this. After then, the code of a method of a low activity degree is destructed.



(19)日本国特許庁（ＪＰ）　(12)公開特許公報（Ａ）　(11)特許出願公開番号
特開2000－222221
（P2000－222221A）
(43)公開日　平成12年８月11日(2000.8.11)

(51)Int.Cl.'	識別記号	FI		テーマコード'(参考)
G06F 9/45		C06F 9/44	322A	5B033
9/42	330	9/42	330A	5B042
11/34		11/34	S	5B081

審査請求　有　請求項の数9　OL　（全８頁）

(21)出願番号　特願平11－22030

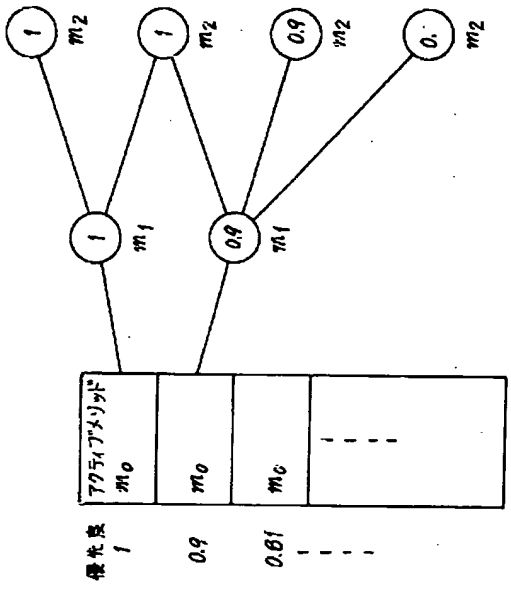
(22)出願日　平成11年１月29日(1999.1.29)

(71)出願人　390009531
インターナショナル・ビジネス・マシーンズ・コーポレ－ション
INTERNATIONAL BUSINESS MASCHINES CORPORATION
アメリカ合衆国10504、ニューヨーク州
アーモンク　（番地なし）
(74)代理人　100086243
弁理士　坂口　博　（外４名）

最終頁に続く

(54)【発明の名称】　コンパイル方法および装置、並びにメソッド活動度計算方法および装置

(57)【要約】
【課題】　コード破棄するメソッドを効果的に選択するためにメソッド活動度を計算する方法を提供する。
【解決の手段】　コンパイル時に、メソッド中で生じ得るメソッド呼び出しに関する呼び出しマップを作成し且つ記憶装置に格納し、実際に生じたメソッド呼び出しを記録するコードを生成し且つ記憶装置に格納する。このような状態において、コードの回収する際には、全スレッドを一時停止し、あるメソッドが実行される可能性の高低を表す活動度を計算する。この時、スタックフレームに対応する第１メソッドについての、呼び出しマップと実際に生じたメソッド呼び出しに関する情報とを用いて、第１メソッドから呼ばれる可能性が高い第２メソッドを特定し且つ記憶装置に格納し、第１メソッドに対応するスタックフレームのスタックの最上部からの段数に応じた優先度を第２メソッドの活動度に作用することにより第２メソッドの活動度を更新する。好適には、このような処理を繰り返す。その後活動度が低いメソッドのコードを破棄する。



ｅｄコードを実行するのが一般的である。コンパイルの最小単位は、メソッドと呼ばれるサブルーチンである。実行頻度の高いコードを機械語に変換することにより、Ｊａｖａバイトコードのマルチプラットフォーム特性を残しつつ、機械語コードの性能を引き出している。

【０００４】メソッドのＪＩＴｅｄコードは、プログラムが動作するＣＰＵに適したコードになっているという点で、Ｃコンパイラなどが生成する最適化コードと同等である。一般に、現在普及しているＣＰＵでは、サブルーチンが呼ばれると、サブルーチンは、自分が使用するローカル変数を格納するための領域（フレームと呼ぶ）をスタック状に形成する。図１は、サブルーチンＡがサブルーチンＢを、サブルーチンＢがサブルーチンＣを呼び出したときのスタックの様子である。この図において、スタックは下から上に伸びる。各領域は、サブルーチンのそれぞれのフレームを表わす。各サブルーチンから戻る際、フレームは除去される。Ｊａｖａでは、スレッドと呼ばれる実行単位毎にＣＰＵ資源が割り振られ、１つのスレッドは固有のスタック（スレッドスタックと呼ぶ）を持つ。ＪＩＴｅｄコードは、スレッドスタック上に上述したようなフレーム（以下、ＪＩＴｅｄフレームと呼ぶ）を作る。

【０００５】メソッドは、最初ほとんどの場合インタプリタで実行されるが、実行頻度あるいは実行時間においてある水準以上実行されると判断されると、ＪＩＴコンパイラによりコンパイルされる。ＪＩＴｅｄコードは一度生成されると、メモリ中に管理される。それらを安全に捨てることができるということＪａｖａシステムが保証する時期は、Ｊａｖａシステムがガベージコレクション（以下、ＧＣ）によってクラスを破棄する（以下、クラスアンロード）ときである。

【０００６】ＪＩＴｅｄコードを含むＪＩＴコンパイラが使用するメモリに制限がある、あるいは制限を設けるとき、クラスアンロードだけでは十分ではない場合がある。例えば、ハードディスクによる仮想記憶を持たない、ネットワークコンピューティング（Ｎｅｔｗｏｒｋ Ｃｏｍｐｕｔｉｎｇ）マシン（以下、NCマシン）のような、シンクライアント（ｔｈｉｎ ｃｌｉｅｎｔ）上で動作するデスクトップ環境では、メモリ制限は現実的な問題である。そのようなデスクトップ環境では、メール、ワープロ、スケジューラ等、複数のＪａｖａアプリケーションが同時に起動され、数千種類のメソッドが呼ばれうるが、かなりの割合のメソッドが各スレッドスタック中に存在しない（アクティブでない）。クラスアンロードは、Ｊａｖａシステム内に完全に使用される可能性のないクラスおよびそのメソッドしか破棄しないため、そうしたアクティブでないメソッドのＪＩＴｅｄコードを破棄しない。そこでＪＩＴコンパイラはＪＩＴｅｄコード破棄メカニズムを持っている。

【０００７】実行中のスレッドの現在のコンテキスト（プログラムカウンタ（ｐｒｏｇｒａｍ ｃｏｕｎｔｅｒ）やスタックポインタ（ｓｔａｃｋ ｐｏｉｎｔｅｒ）などＣＰＵ資源のコピー）あるいはそのスレッドスタック中に保持されるＪＩＴｅｄコードのフレームには、そのスレッドでのアクティブなメソッドが記録されている。アクティブなメソッドのＪＩＴｅｄコードが破棄されると、スレッドが実行できない。したがって、破棄するのは、非アクティブなＪＩＴｅｄコードである。ＪＩＴｅｄコード破棄メカニズムにおいて最も簡単な破棄の仕方は、アクティブでないＪＩＴｅｄコードをすべて破棄する方法である。しかし、この方法では必要以上にＪＩＴｅｄコードを破棄してしまい、無駄である。メソッドの最適化コンパイルには時間がかかるため、「すぐに実行されるメソッドを破棄しない」という方針の下で選択的に破棄すべきである。的確な選択の効果の例として、ＪＩＴｅｄコード破棄が今行われ、すぐに実行されるメソッドＡと、１０分後に実行されるメソッドＢがあったとする。もしＡを破棄すると、今破棄したＡのＪＩＴｅｄコードのメモリ量と、さらにＡのコンパイルに必要なワークのメモリ量とを、直ちに確保しなければならない。一方、Ｂを破棄すれば、１０分間はＢのＪＩＴｅｄコードのメモリ量は、利用可能メモリとなる。実際には必要なフリーメモリ量分の複数のＪＩＴｅｄコードを破棄する。理想的な破棄の方針は、今後使用されないＪＩＴｅｄコードを破棄することであるが、このような情報は普通得られない。類推し、しばらく呼ばれそうにない順にＪＩＴｅｄコードを破棄するのが現実的である。以下、これをＪＩＴｅｄコード破棄方針と呼ぶ。

【０００８】通常考えられるメソッドのプロファイルに、実行頻度（何回呼ばれたか）と実行時間（プログラムカウンタがそのメソッドを指していた時間）がある。実行頻度の収集は、メソッドが呼ばれる毎にそのメソッドのカウンタを増加させて行なう。メソッドがＪＩＴコンパイラによるコンパイルされると、それまでインタプリタで測定されていた実行頻度の更新は終了し、したがってインタプリタからはＪＩＴｅｄコードに対する実行頻度情報は得られない。ＪＩＴｅｄコードで実行頻度情報を得るには、各メソッドのＪＩＴｅｄコードのプロローグで実行回数を数える必要がある。例えば、ｘ８６プラットフォームでは、inc dword ptr ［カウンタアドレス］命令を追加する。実行時間の収集は、一定時間毎に割り込み（典型的にはタイマ割り込み、プロセッサによっては実行命令数割り込み）をかけ、プログラムカウンタ（以下、ｐｃ）を収集して行なう。

【０００９】ＪＩＴｅｄコード破棄方針として、これら従来のプロファイルを使用すると問題が生じる。まず、実行頻度について、各メソッドのＪＩＴｅｄコード実行の度にメモリを参照してカウントするのは、メモリが枯渇しない状態へのオーバヘッドになり、問題である。さらに、頻度が少ないが直ちに呼ばれるメソッドを誤って

破棄してしまう恐れがある。知りたいのは、呼ばれずにメモリを占めているＪＩＴｅｄコードであるから、目的に合わない。また実行時間についても、割り込みと収集したｐｃの管理（すぐにメソッドに対応づけるなら対応づけのコスト、対応づけないならその保存）は、メモリが枯渇しない状態へのオーバヘッドになり問題である。さらに実行頻度収集の場合と同様、実行時間が短いが直ちに呼ばれるメソッドを誤って破棄してしまう恐れがある。

【００１０】

【発明が解決しようとする課題】上述のように、従来のプロファイル手法をＪＩＴｅｄコード破棄方針とするのは的外れで、不適切なＪＩＴｅｄコード破棄が起きる可能性がある。本発明の目的は、コード破棄するメソッドを効果的に選択するために、メソッドがすぐに実行されやすいことを表わすメソッド活動度を計算する方法を提供することである。

【００１１】

【課題を解決するための手段】本発明は上記課題を解決するため、コンパイル時に、メソッド中で生じ得るメソッド呼び出しに関する呼び出しマップを作成し且つ記憶装置に格納し、実際に生じたメソッド呼び出しを記録するコードを生成し且つ記憶装置に格納する。この呼び出しマップは、好適には、メソッド中のあるアドレスに対し、当該アドレス以降に生じ得るメソッド呼び出しを出力するように構成される。但し、メソッドごとにメソッド呼び出しを出力するようにしてもよい。コンパイルが終了し且つ生成されたコードが実行されると、メソッド中で生じ得るメソッド呼び出しに関する呼び出しマップと実際に生じたメソッド呼び出しに関する情報とが各メソッドごとに記録されるようになる。

【００１２】このような状態において、コードを回収する際には、全スレッドを一時停止し、あるメソッドが実行される可能性の高低を表す活動度を計算する。この時、スタックフレームに対応する第１メソッドについての、呼び出しマップと実際に生じたメソッド呼び出しに関する情報とを用いて、第１メソッドから呼ばれる可能性が高い第２メソッドを特定し且つ記憶装置に格納し、第１メソッドに対応するスタックフレームのスタックの最上部からの段数に応じた優先度を第２メソッドの活動度に作用することにより第２メソッドの活動度を更新する。好適には、このような処理を繰り返す。すなわち、第２メソッドについての、呼び出しマップと実際に生じたメソッド呼び出しに関する情報と用いて、第２メソッドから呼ばれる可能性が高い第３メソッドを特定し且つ記憶装置に格納し、第１メソッドに対応するスタックフレームのスタックの最上部からの段数に応じた優先度を第３メソッドの活動度に作用することにより第３メソッドの活動度を更新する。このようにして求められたメソッドの活動度が低いもののコードを破棄するとより効率的なコード破棄が行える。

【００１３】なお、好適には、呼び出しマップは、メソッド中のあるアドレスに対し、当該アドレス以降に生じ得るメソッド呼び出しを出力し、第２メソッドの特定は、第１メソッドの実行再開アドレス以降に生じ得るメソッド呼び出しを用いて行われる。

【００１４】本発明では、非アクティブメソッドについて、現在のコンテキスト、プログラムの構造、実行時情報を利用して、すぐに呼ばれる度合い（活動度）を求める。ここでいう、現在のコンテキストの利用とは、スタック上位（トップに近い）にあるメソッドほど早く実行されそうだと判断できることである。プログラムの構造の利用とは、コントロールフロー上、一時停止しているメソッドから今後呼ばれそうなメソッドを知ることである。実行時情報の利用とは、前記呼ばれそうなメソッドでも（インタプリタを含めた）実行時に一度も呼ばれていなければ、判断対象から除外できるということである。早く呼ばれそうなメソッドの活動度が、その呼ばれる早さに応じて上げられる。

【００１５】このように活動度は、メソッドの再実行時刻が早いほど、その値が大きい。これは、上述したＪＩＴｅｄコード破棄方針をよく模倣する点で優れている。先ほどの例でいえば、仮にＡがスタック上位、Ｂがスタック下位からそれぞれ呼ばれるメソッドであれば、本発明のアルゴリズムによってＡの活動度はＢの活動度より高くなり、Ｂより先にＡを選んで破棄することはなくなる。

【００１６】あるスレッドでＪＩＴコンパイラのメモリ要求が満たせなくなったとき、そのメソッドは自分以外のスレッドをすべて一時停止し、アクティブメソッドを見つける。本発明によるメソッド活動度計算方法により、各メソッドの活動度を決定する。その後、このような活動度に基づいて、非アクティブなメソッドの一部あるいは全部のＪＩＴｅｄコードを破棄する。一時停止したスレッドをすべて再開し、メモリ要求を再度行なう。これをメモリ要求が満たされるまで行なう。最終的にメモリ要求が満たされない場合、許されるなら、コンパイラの最適化レベルを下げて再度コンパイルに挑戦する。再度のコンパイルが許されない場合、あるいは最低の最適化レベルでもコンパイルできなかった場合は、このメソッドをコンパイルしない。

【００１７】メソッド活動度を計算するための作業コストを低減するために、メソッドの活動度を、これらがさらに呼び出すメソッドの活動度に作用させる動作の回数に上限を設けてもよい。

【００１８】

【発明の実施の形態】本発明は、ＪＩＴｅｄコード破棄方針を模倣するために、メソッドに活動度と呼ばれる優先順位を与える方法である。ＪＩＴｅｄコード破棄方針とは、「しばらく呼ばれそうにない順にＪＩＴｅｄコー

ドを破棄する」ことである。低い活動度のメソッドほど、しばらく呼ばれないと予想できる。しばらく呼ばれないメソッドのＪＩＴｅｄコードを破棄すれば、それらによって使われていたメモリ量のフリーメモリは長い間活用できるはずである。

【００１９】以下、メソッドmの活動度Ａ（ｍ）の計算手順を示す。各メソッドの活動度の初期値を０とする。まず、ＪＩＴコンパイラによるコンパイルおよび実行時における動作を、図２のフローチャートを参照して説明する。ステップＳ２０１で開始し、ステップＳ２０２で、メソッドmに関して、ＪＩＴｅｄコード中のアドレスａｄｄｒと、このアドレスからメソッドの実行が再開したときに呼ばれる（あるいは可能性がある）メソッド集合とを対応させるための、アドレス－呼び出しマップＭＡＰＡ－Ｉ（ｍ）を作る。アドレス－呼び出しマップは、ａｄｄｒを含む基本ブロックＢＢ（ｂａｓｉｃ　ｂｌｏｃｋ）と、このＢＢからコントロールフロー上で到達する前記メソッド呼び出しとを対応づけるものである。このアドレス－呼び出しマップは、あるアドレスａｄｄｒが与えられたときに、このａｄｄｒから到達する全メソッド呼び出しを返す。この全メソッド呼び出しを呼び出し集合ＳＥＴＡ－Ｉ（ｍ，ａｄｄｒ）と呼ぶ。

【００２０】アドレス－呼び出しマップの計算をするためには、コントロールフローを解析し、基本ブロックＢＢ毎に、ＢＢが到達する可能性のあるメソッド呼び出しのアドレスを集める。それから、ＢＢの先頭アドレスi _addrとリーチされるメソッド呼び出しアドレスi _callの関係データ（例えば、（i _addr，i _call）をソートした表）をコンパイルコードにつける。呼び出し集合は、ａｄｄｒを含むＢＢに対応するすべてのi _call である。図3は、メソッドＥのＪＩＴｅｄコードを示す線図である。メソッドＥは、ＢＢ１ないしＢＢ５から成る。この図において、アドレス１１５において、メソッドＦを呼び出している。したがって、呼び出し集合は、ａｄｄｒ＝１００～１１５はＦを返し、ａｄｄｒ＝１１６～１２０は何も返さない。

【００２１】アドレス－呼び出しマップの計算コストを大幅に削減するには、正確でないが、メソッドmのアドレス－呼び出しマップを、mに含まれるすべてのメソッド呼び出しの集合としてもよい。このとき呼び出し集合もmに含まれるすべてのメソッド呼び出しの集合である。

【００２２】ＪＩＴｅｄコード中にあるメソッド呼び出しのためのｃａｌｌ命令は、最適化のために、なるべくダイレクトコール（direct call ）になるように処理される。例えば、あるメソッド呼び出しをコンパイルする際に呼び出し先のメソッドがまだコンパイルされていない場合、コンパイラはダイレクトコール命令を生成できない。このようなメソッド呼び出しが発生すると、バックパッチコードにジャンプして、ＪＩＴコンパイラによって生成されたターゲットアドレスを次回のメソッド呼び出しから用いることができるように設定する。このような処理は従来から行われてきた。本発明では、メソッド呼び出しを最初に行う場合には、ターゲットアドレスがまだ生成されていない場合でも生成されている場合でも必ず、バックパッチコードにジャンプし、ターゲットアドレスを記録するようにする（Ｓ２０３のコードを用いる）。２回目以降のメソッド呼び出しでは、バックパッチコードにジャンプする必要はない。

【００２３】ステップＳ２０３で、メソッド呼び出しのバックパッチコード中に、各呼び出しコードによって実際に呼び出しが起きたことを記録するコードを生成する。この記録を有効呼び出し集合ＳＥＴＥＩ（ｍ）と呼ぶ。この有効呼び出し集合は、呼び出しアドレスとターゲットアドレスのペア（ei_call,ei _target）の集合である。ei_targetの初期値はＮＵＬＬである。ei_targetがＮＵＬＬでない要素が意味のある要素である。

【００２４】ステップＳ２０４で、コンパイラは、メソッドをコンパイルする際に、全てのターゲットアドレスについての表（ei_call，ei_target）を作り、コンパイルコードに付ける。例えば、表１のようなものである。

【表１】

ei_call	ei_target
xxxx	null
yyyy	null
zzzz	null
・ ・	・ ・

この表が有効呼び出し集合となる。コンパイル時の処理は以上である。

【００２５】ＪＩＴｅｄコードが実行され、Ｓ２０３で生成されたコードが実行されると、呼び出しを行なったメソッドの有効呼び出し集合の対応するエントリ（ei_call，ei_target）のei_targetが更新される。ここで、有効呼び出し集合においてei_targetに実際のターゲットアドレスが入っている呼び出しのみが、実際に呼び出しを行なう有効呼び出しである。ステップＳ２０６で処理を終了する。なお、表１は、例えば表２のようになる。

【表２】

Tコンパイラ５６を含む。ＪａｖａＶＭ５２は、図示しないガベージコレクタやスタックトレーサをさらに含む。なお、クライアント・コンピュータ５は、通常のコンピュータの他、メモリの大きさが小さかったり、ハードディスク等の補助記憶装置を含まないような、いわゆるネットワーク・コンピュータや情報家電の場合もある。

【００４０】サーバ・コンピュータ１では、Ｊａｖａソースコード１０は、Ｊａｖａコンパイラ１２によりコンパイルされる。このコンパイルの結果が、バイトコード１４である。このバイトコード１４は、ネットワーク３を介してクライアント・コンピュータ５に送信される。バイトコード１４は、クライアント・コンピュータ５内のWWWブラウザ（World Wide Web Browser）などに設けられたＪａｖａ仮想マシン（Java VM ）５２にとってネイティブ・コードであり、実際ハードウエア５５のＣＰＵにて実行する場合には、Ｊａｖａインタープリタ５４や、ＪａｖａＪＩＴコンパイラ５６を用いる。インタープリタ５４は、実行時にバイトコード１４をデコードし、命令ごとに用意される処理ルーチンを呼び出して実行する。一方、ＪＩＴコンパイラ５６は、バイトコードを事前にあるいは実行する直前にコンパイラを用いてマシン・コード５８に変換してそれをＣＰＵで実行する。

【００４１】

【発明の効果】本発明によれば、コード破棄するメソッドを効果的に選択することができるため、同一メソッドの再コンパイル回数が減少し、コンパイルオーバヘッドを減少させることができる。

【図面の簡単な説明】

【図１】　サブルーチンＡがサブルーチンＢを、サブルーチンＢがサブルーチンＣを呼び出したときのスタックを示す線図である。

【図２】　本発明によるメソッド活動度計算方法のＪＩＴコンパイルおよび実行時における動作を説明するフローチャートである。

【図３】　メソッドＥのＪＩＴｅｄコードを示す線図である

【図４】　本発明によるメソッド活動度計算方法のＪＩＴｅｄコードの回収プロセス中における動作を説明するフローチャートである。

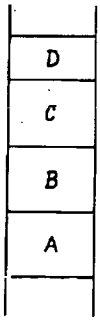
【図５】　ターゲットメソッドへの活動度の割り当てを説明する線図である。

【図６】　本発明における装置構成の一例を示す図である。

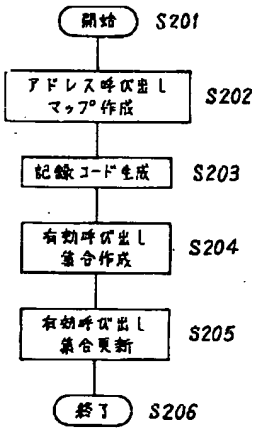
【符号の説明】

１　サーバ・コンピュータ
３　ネットワーク
５　クライアント・コンピュータ
１０　Ｊａｖａソースコード
１２　Ｊａｖａコンパイラ
１４　バイトコード
５２　ＪａｖａＶＭ
５４　Ｊａｖａインタプリタ
５６　Ｊａｖａ　ＪＩＴコンパイラ
５８　マシンコード
６０　ガベージ・コレクタ
５３　ＯＳ
５５　ハードウエア（ＣＰＵ及びメモリを含む）

【図１】



【図２】



【図５】